In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```python
#load data file
data = pd.read_csv('D:\Semester 6 - 3rd year\Machine Learning -CO544\data.csv')
```

In [3]:
```python
data.head()
```

Out[3]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	u	g	0.00	w	0	True	v	1.25	True	1	False	202	g	Success
1	a	58.67	u	g	4.46	q	560	True	h	3.04	True	6	False	43	g	Success
2	a	24.5	u	g	0.50	q	824	False	h	1.50	True	0	False	280	g	Success
3	b	27.83	u	g	1.54	w	3	True	v	3.75	True	5	True	100	g	Success
4	b	25	u	g	11.25	c	1208	True	v	2.50	True	17	False	200	g	Success

In [4]:
```
#find number of missing data in each column
data.__eq__('?').sum()
```

```
c:\python\python38\lib\site-packages\pandas\core\ops\array_ops.py:253: FutureWarning: elementwise comparison failed; r
eturning scalar instead, but in the future will perform elementwise comparison
  res_values = method(rvalues)
```

Out[4]:
```
A1      8
A2     10
A3      4
A4      4
A5      0
A6      6
A7      0
A8      0
A9      6
A10     0
A11     0
A12     0
A13     0
A14    10
A15     0
A16     0
dtype: int64
```

In [5]:
```
#replace missing data in each column with nan
data['A1'].replace('?',np.nan, inplace=True)
data['A2'].replace('?',np.nan, inplace=True)
data['A3'].replace('?',np.nan, inplace=True)
data['A4'].replace('?',np.nan, inplace=True)
data['A6'].replace('?',np.nan, inplace=True)
data['A9'].replace('?',np.nan, inplace=True)
data['A14'].replace('?',np.nan, inplace=True)
```

In [6]:
```
data.describe()
```

Out[6]:

	A5	A7	A10	A12
count	552.000000	552.000000	552.000000	552.000000
mean	4.884384	1100.827899	2.398678	2.614130
std	5.086809	5628.306468	3.551266	5.161073
min	0.000000	0.000000	0.000000	0.000000
25%	1.083750	0.000000	0.165000	0.000000
50%	2.750000	5.000000	1.000000	0.000000
75%	7.551250	456.500000	3.000000	3.000000
max	28.000000	100000.000000	28.500000	67.000000

In [7]:
```
#drop rows with missing data
data.dropna()
```

Out[7]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	u	g	0.000	w	0	True	v	1.250	True	1	False	202	g	Success
1	a	58.67	u	g	4.460	q	560	True	h	3.040	True	6	False	43	g	Success
2	a	24.5	u	g	0.500	q	824	False	h	1.500	True	0	False	280	g	Success
3	b	27.83	u	g	1.540	w	3	True	v	3.750	True	5	True	100	g	Success
4	b	25	u	g	11.250	c	1208	True	v	2.500	True	17	False	200	g	Success
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
547	b	39.17	u	g	1.625	c	4700	True	v	1.500	True	10	False	186	g	Success
548	b	39.08	u	g	6.000	m	1097	True	v	1.290	True	5	True	108	g	Success
549	b	31.67	u	g	0.830	x	3290	True	v	1.335	True	8	True	303	g	Success
550	b	41	u	g	0.040	e	0	True	v	0.040	False	1	False	560	s	Success
551	b	48.5	u	g	4.250	m	0	False	v	0.125	True	0	True	225	g	Success

524 rows × 16 columns

In [8]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 552 entries, 0 to 551
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      544 non-null    object 
 1   A2      542 non-null    object 
 2   A3      548 non-null    object 
 3   A4      548 non-null    object 
 4   A5      552 non-null    float64
 5   A6      546 non-null    object 
 6   A7      552 non-null    int64  
 7   A8      552 non-null    bool   
 8   A9      546 non-null    object 
 9   A10     552 non-null    float64
 10  A11     552 non-null    bool   
 11  A12     552 non-null    int64  
 12  A13     552 non-null    bool   
 13  A14     542 non-null    object 
 14  A15     552 non-null    object 
 15  A16     552 non-null    object 
dtypes: bool(3), float64(2), int64(2), object(9)
memory usage: 57.8+ KB
None
```

In [9]:
```python
data = data.dropna()
```

In [10]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 524 entries, 0 to 551
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      524 non-null    object 
 1   A2      524 non-null    object 
 2   A3      524 non-null    object 
 3   A4      524 non-null    object 
 4   A5      524 non-null    float64
 5   A6      524 non-null    object 
 6   A7      524 non-null    int64  
 7   A8      524 non-null    bool   
 8   A9      524 non-null    object 
 9   A10     524 non-null    float64
 10  A11     524 non-null    bool   
 11  A12     524 non-null    int64  
 12  A13     524 non-null    bool   
 13  A14     524 non-null    object 
 14  A15     524 non-null    object 
 15  A16     524 non-null    object 
dtypes: bool(3), float64(2), int64(2), object(9)
memory usage: 58.8+ KB
None
```

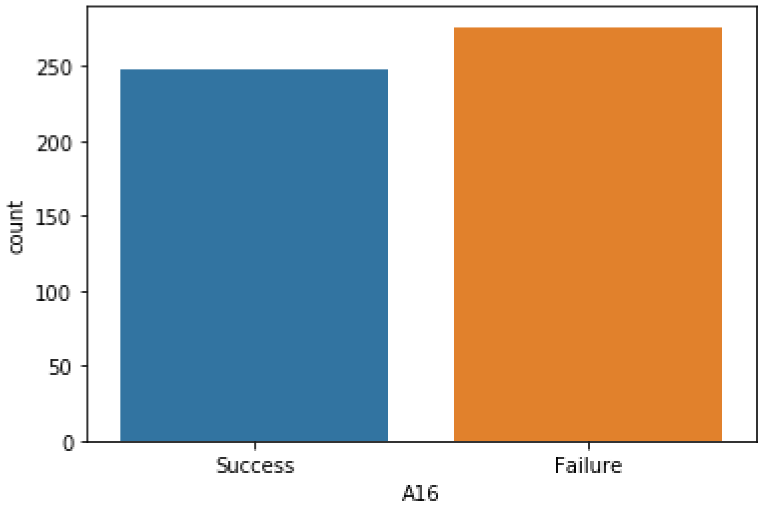
In [11]:
```
#change object type to float type of some columns
data["A2"]= data["A2"].astype(float)
data["A12"]= data["A12"].astype(float)
data["A7"]= data["A7"].astype(float)
data["A14"]= data["A14"].astype(float)
```

In [12]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 524 entries, 0 to 551
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      524 non-null    object 
 1   A2      524 non-null    float64
 2   A3      524 non-null    object 
 3   A4      524 non-null    object 
 4   A5      524 non-null    float64
 5   A6      524 non-null    object 
 6   A7      524 non-null    float64
 7   A8      524 non-null    bool   
 8   A9      524 non-null    object 
 9   A10     524 non-null    float64
 10  A11     524 non-null    bool   
 11  A12     524 non-null    float64
 12  A13     524 non-null    bool   
 13  A14     524 non-null    float64
 14  A15     524 non-null    object 
 15  A16     524 non-null    object 
dtypes: bool(3), float64(6), object(7)
memory usage: 58.8+ KB
None
```

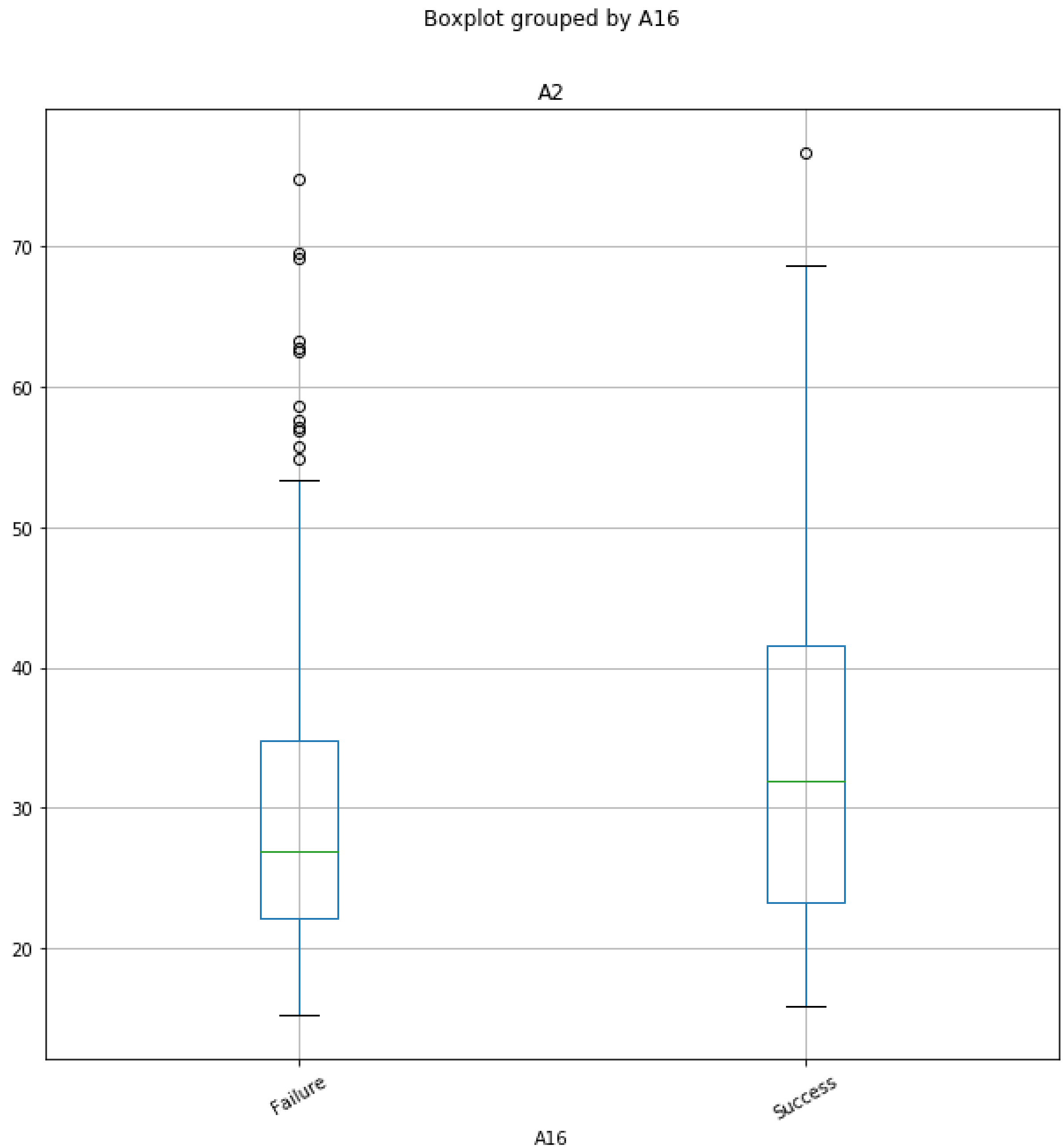
In [13]:
```
import seaborn as sns
```

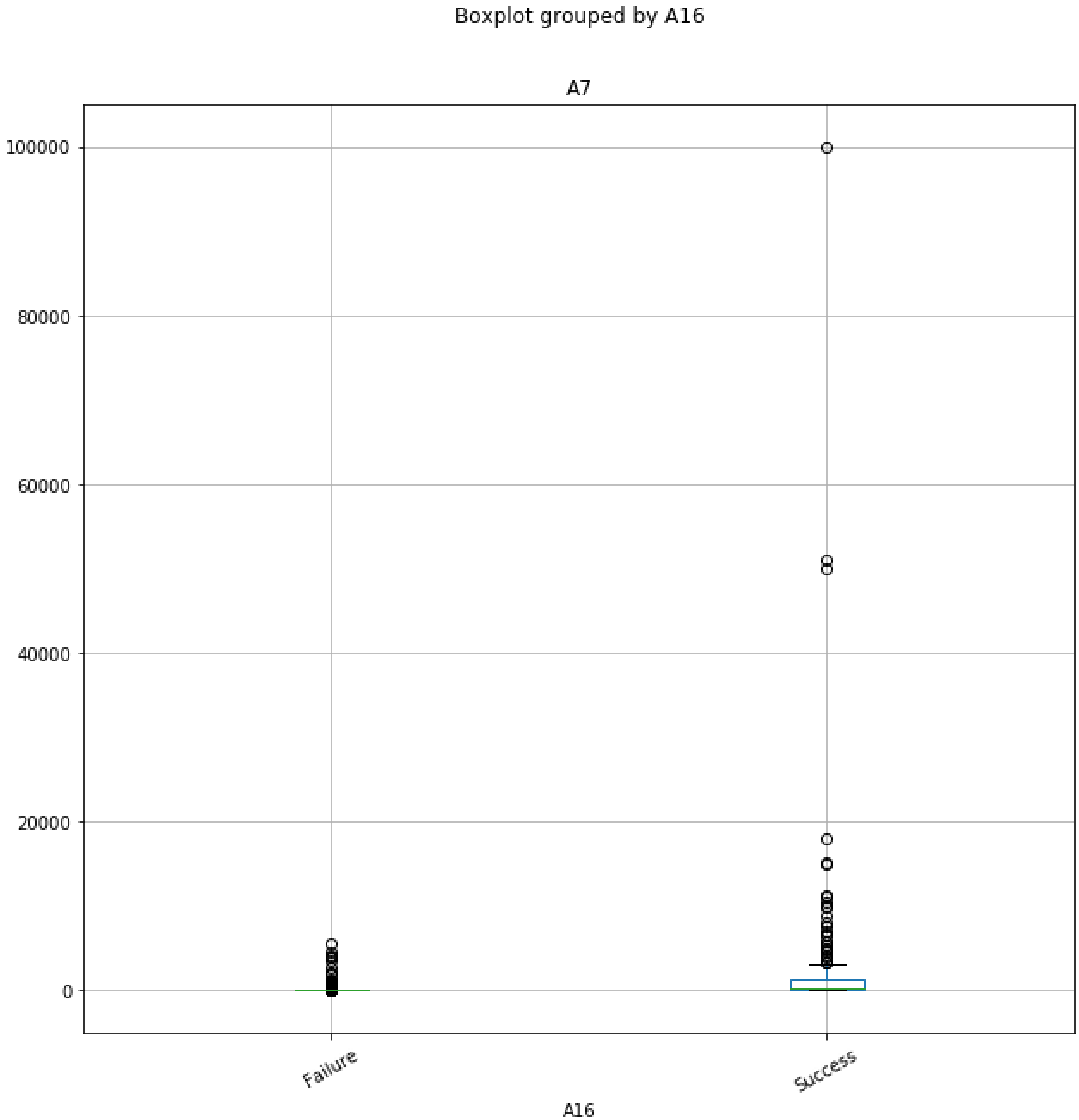
In [14]:
```
sns.countplot(data['A16'],label="count")
plt.show()
```

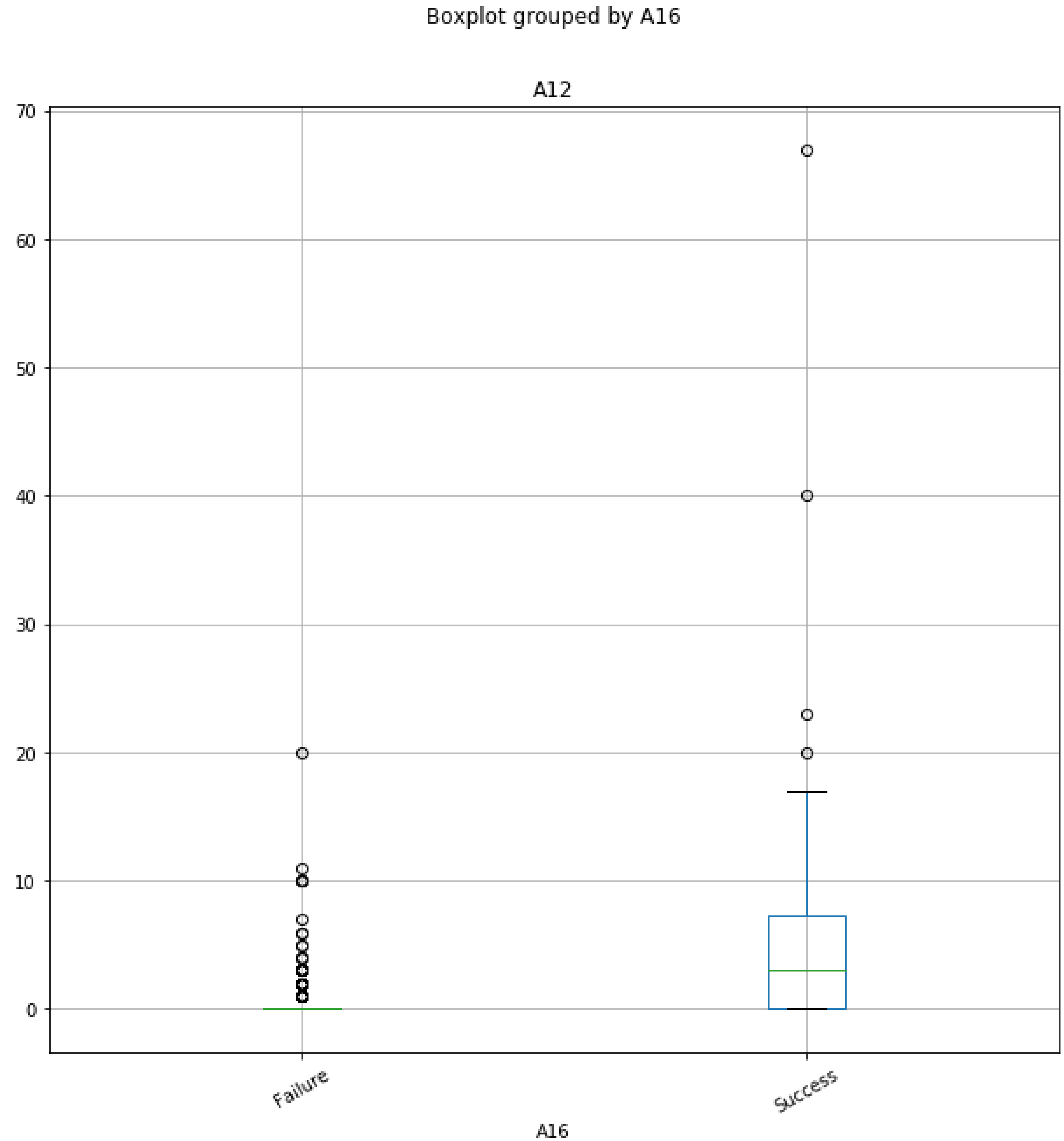


In [15]:
```
data.boxplot('A2','A16',rot=30, figsize=(10,10))
data.boxplot('A7','A16',rot=30, figsize=(10,10))
data.boxplot('A12','A16',rot=30, figsize=(10,10))
data.boxplot('A14','A16',rot=30, figsize=(10,10))
```

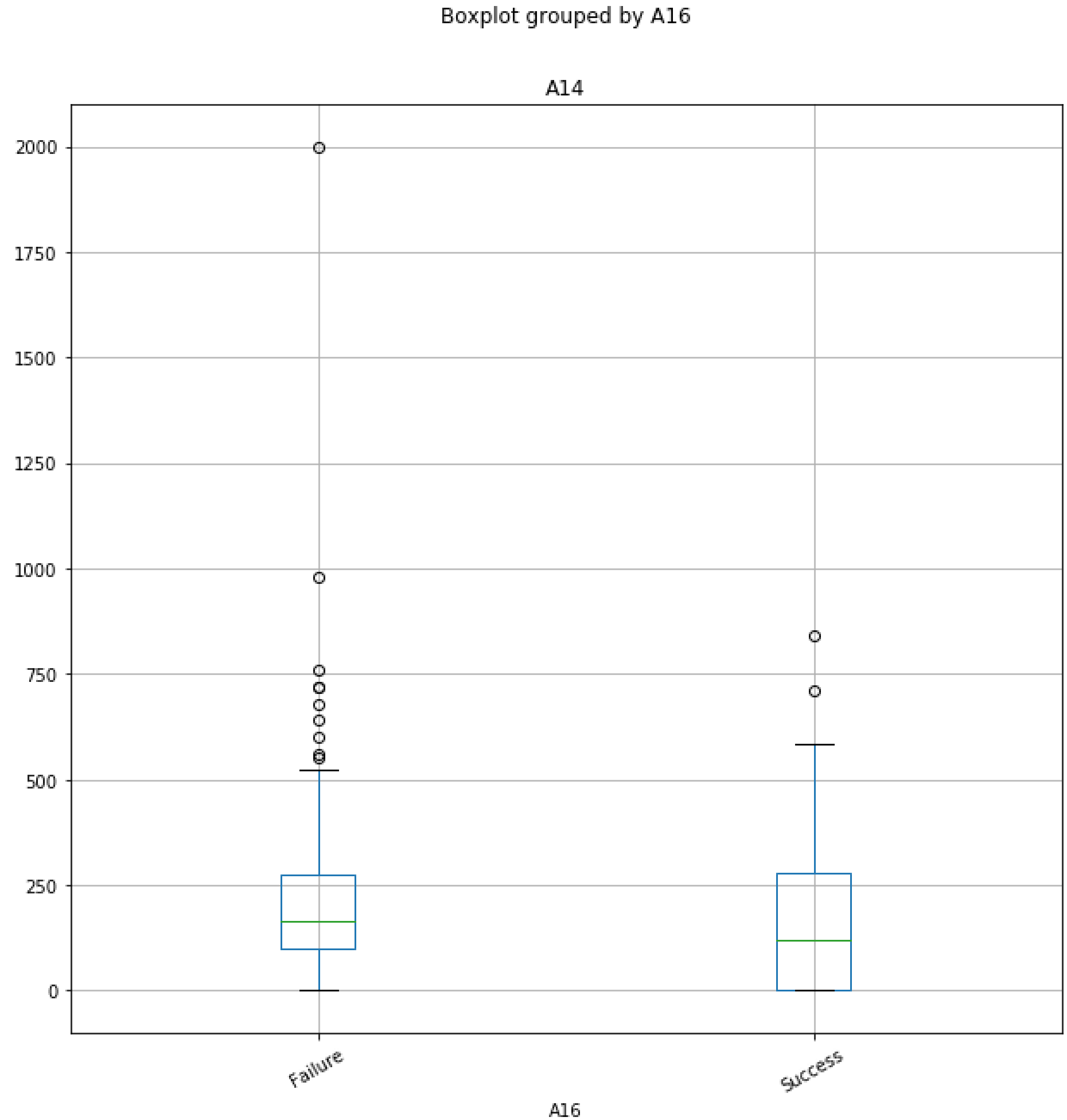
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x201be5ce7f0>




Boxplot grouped by A16
A7
100000
80000
60000
40000
20000
0
Failure
Success
A16



Boxplot grouped by A16
A12
70
60
50
40
30
20
10
0
Failure
Success
A16



In [16]:
```
newer_data = data.copy()
```

In [17]:

```
print(newer_data['A1'].value_counts())
print(newer_data['A3'].value_counts())
print(newer_data['A4'].value_counts())
print(newer_data['A6'].value_counts())
print(newer_data['A9'].value_counts())
print(newer_data['A15'].value_counts())
```

```
b    365
a    159
Name: A1, dtype: int64
u    398
y    124
l      2
Name: A3, dtype: int64
g     398
p     124
gg      2
Name: A4, dtype: int64
c     100
q      65
w      50
i      47
ff     40
aa     39
k      34
x      33
cc     32
m      30
e      22
d      21
j       9
r       2
Name: A6, dtype: int64
v     296
h     113
bb     48
ff     43
z       7
j       6
dd      6
n       3
o       2
Name: A9, dtype: int64
g    476
s     46
p      2
Name: A15, dtype: int64
```

In [18]:

```
#one-hot encoding to object type columns
onehote_data = newer_data.copy()
onehote_data = pd.get_dummies(onehote_data, columns=['A3'], prefix=['A3'])
onehote_data = pd.get_dummies(onehote_data, columns=['A4'], prefix=['A4'])
onehote_data = pd.get_dummies(onehote_data, columns=['A6'], prefix=['A6'])
onehote_data = pd.get_dummies(onehote_data, columns=['A9'], prefix=['A9'])
onehote_data = pd.get_dummies(onehote_data, columns=['A15'], prefix=['A15'])
```

In [19]:

```
onehote_data.head()
```

Out[19]:

	A1	A2	A5	A7	A8	A10	A11	A12	A13	A14	...	A9_ff	A9_h	A9_j	A9_n	A9_o	A9_v	A9_z	A15_g	A15_p	A15_s
0	b	30.83	0.00	0.0	True	1.25	True	1.0	False	202.0	...	0	0	0	0	0	1	0	1	0	0
1	a	58.67	4.46	560.0	True	3.04	True	6.0	False	43.0	...	0	1	0	0	0	0	0	1	0	0
2	a	24.50	0.50	824.0	False	1.50	True	0.0	False	280.0	...	0	1	0	0	0	0	0	1	0	0
3	b	27.83	1.54	3.0	True	3.75	True	5.0	True	100.0	...	0	0	0	0	0	1	0	1	0	0
4	b	25.00	11.25	1208.0	True	2.50	True	17.0	False	200.0	...	0	0	0	0	0	1	0	1	0	0

5 rows × 43 columns

In [20]:

```
#label encoding to column A1
onehote_data["A1"]= onehote_data["A1"].astype('category')
onehote_data['A1'] = onehote_data['A1'].cat.codes
onehote_data.head()
```

Out[20]:

	A1	A2	A5	A7	A8	A10	A11	A12	A13	A14	...	A9_ff	A9_h	A9_j	A9_n	A9_o	A9_v	A9_z	A15_g	A15_p	A15_s
0	1	30.83	0.00	0.0	True	1.25	True	1.0	False	202.0	...	0	0	0	0	0	1	0	1	0	0
1	0	58.67	4.46	560.0	True	3.04	True	6.0	False	43.0	...	0	1	0	0	0	0	0	1	0	0
2	0	24.50	0.50	824.0	False	1.50	True	0.0	False	280.0	...	0	1	0	0	0	0	0	1	0	0
3	1	27.83	1.54	3.0	True	3.75	True	5.0	True	100.0	...	0	0	0	0	0	1	0	1	0	0
4	1	25.00	11.25	1208.0	True	2.50	True	17.0	False	200.0	...	0	0	0	0	0	1	0	1	0	0

5 rows × 43 columns

In [21]:

```
print(onehote_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 524 entries, 0 to 551
Data columns (total 43 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      524 non-null    int8   
 1   A2      524 non-null    float64
 2   A5      524 non-null    float64
 3   A7      524 non-null    float64
 4   A8      524 non-null    bool   
 5   A10     524 non-null    float64
 6   A11     524 non-null    bool   
 7   A12     524 non-null    float64
 8   A13     524 non-null    bool   
 9   A14     524 non-null    float64
 10  A16     524 non-null    object 
 11  A3_l    524 non-null    uint8  
 12  A3_u    524 non-null    uint8  
 13  A3_y    524 non-null    uint8  
 14  A4_g    524 non-null    uint8  
 15  A4_gg   524 non-null    uint8  
 16  A4_p    524 non-null    uint8  
 17  A6_aa   524 non-null    uint8  
 18  A6_c    524 non-null    uint8  
 19  A6_cc   524 non-null    uint8  
 20  A6_d    524 non-null    uint8  
 21  A6_e    524 non-null    uint8  
 22  A6_ff   524 non-null    uint8  
 23  A6_i    524 non-null    uint8  
 24  A6_j    524 non-null    uint8  
 25  A6_k    524 non-null    uint8  
 26  A6_m    524 non-null    uint8  
 27  A6_q    524 non-null    uint8  
 28  A6_r    524 non-null    uint8  
 29  A6_w    524 non-null    uint8  
 30  A6_x    524 non-null    uint8  
 31  A9_bb   524 non-null    uint8  
 32  A9_dd   524 non-null    uint8  
 33  A9_ff   524 non-null    uint8  
 34  A9_h    524 non-null    uint8  
 35  A9_j    524 non-null    uint8  
 36  A9_n    524 non-null    uint8  
 37  A9_o    524 non-null    uint8  
 38  A9_v    524 non-null    uint8  
 39  A9_z    524 non-null    uint8  
 40  A15_g   524 non-null    uint8  
 41  A15_p   524 non-null    uint8  
```

```
 42  A15_s    524 non-null    uint8
dtypes: bool(3), float64(6), int8(1), object(1), uint8(32)
memory usage: 51.2+ KB
None
```

In [22]:

```
feature_names = ['A1', 'A2', 'A5', 'A7','A8','A10','A11','A12','A13','A14','A3_l','A3_u','A3_y','A4_g','A4_gg','A4_p',
'A6_aa','A6_c','A6_cc','A6_d','A6_e','A6_ff','A6_i','A6_j','A6_k','A6_m','A6_q','A6_r','A6_w','A6_x','A9_bb','A9_dd','A
9_ff','A9_h','A9_j','A9_n','A9_o','A9_v','A9_z','A15_g','A15_p','A15_s']
X = onehote_data[feature_names]
Y = onehote_data['A16']
```

In [23]:

```
#split the data set as training set and test set randomly
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)
```

In [24]:

```
#apply scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [25]:

```
#use model Logistic Regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X_train, y_train)

print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(logreg.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(logreg.score(X_test, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.85
Accuracy of Logistic regression classifier on test set: 0.94
```

In [26]:
```
#use model Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier().fit(X_train, y_train)

print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.86
```

In [27]:
```
#use model Decision Tree Classifier with maximum depth of 3
clf2 = DecisionTreeClassifier(max_depth=3).fit(X_train, y_train)
print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf2.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf2.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 0.82
Accuracy of Decision Tree classifier on test set: 0.93
```

In [28]:
```
#use model Decision Tree Classifier with maximum depth of 4
clf2 = DecisionTreeClassifier(max_depth=4).fit(X_train, y_train)
print('Accuracy of Decision Tree classifier on training set: {:.2f}'
     .format(clf2.score(X_train, y_train)))
print('Accuracy of Decision Tree classifier on test set: {:.2f}'
     .format(clf2.score(X_test, y_test)))
```

```
Accuracy of Decision Tree classifier on training set: 0.87
Accuracy of Decision Tree classifier on test set: 0.90
```

In [29]:
```
#use model k-neighbours
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'
     .format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.85
Accuracy of K-NN classifier on test set: 0.89
```

In [30]:
```
#use model Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
print('Accuracy of LDA classifier on training set: {:.2f}'
     .format(lda.score(X_train, y_train)))
print('Accuracy of LDA classifier on test set: {:.2f}'
     .format(lda.score(X_test, y_test)))
```

```
Accuracy of LDA classifier on training set: 0.87
Accuracy of LDA classifier on test set: 0.95
```

In [31]:
```
#use model Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)
print('Accuracy of GNB classifier on training set: {:.2f}'
     .format(gnb.score(X_train, y_train)))
print('Accuracy of GNB classifier on test set: {:.2f}'
     .format(gnb.score(X_test, y_test)))
```

```
Accuracy of GNB classifier on training set: 0.61
Accuracy of GNB classifier on test set: 0.59
```

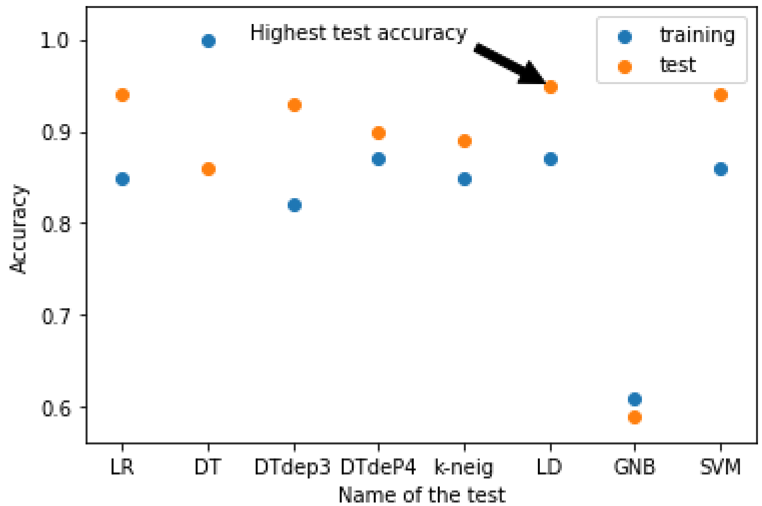
In [32]:
```
#use model support vector machine
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)
print('Accuracy of SVM classifier on training set: {:.2f}'
     .format(svm.score(X_train, y_train)))
print('Accuracy of SVM classifier on test set: {:.2f}'
     .format(svm.score(X_test, y_test)))
```

```
Accuracy of SVM classifier on training set: 0.86
Accuracy of SVM classifier on test set: 0.94
```

In [51]:
```
Test_names = ['Logistic regression', 'Decision Tree', 'Decision Tree-max depth=3', 'Decision Tree-max depth=4', 'k-neig
hbors','Linear Discriminant','Gaussian Naive Bayes', 'Support vector machine']
Test_name_codes= ['LR', 'DT', 'DTdep3', 'DTdeP4', 'k-neig','LD','GNB', 'SVM']
training_accuracy = [0.85,1,0.82,0.87,0.85,0.87,0.61,0.86]
test_accuracy = [0.94,0.86,0.93,0.90,0.89,0.95,0.59,0.94]
```

In [52]:
```
plt.scatter(Test_name_codes,training_accuracy,label='training')
plt.scatter(Test_name_codes,test_accuracy,label='test')
plt.xlabel('Name of the test')
plt.ylabel('Accuracy')
plt.annotate('Highest test accuracy', xy=('LD', 0.95), xytext=(1.5, 1),
             arrowprops=dict(facecolor='black', shrink=0.05),
             )
plt.legend()
plt.show()
```



In [35]:
```
#use linear discriminant since it has the highest accuracy in test set and has second highest accuracy when considering the training test
#then select the 'lda' model
```

In [36]:
```
original_test_data = pd.read_csv('D:/Semester 6 - 3rd year/Machine Learning -CO544/testdata_10%.csv')
test_data = original_test_data.copy()
test_data.head()
```

Out[36]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
0	b	32.67	y	p	9.000	w	0	False	h	5.250	True	0	True	154	g
1	?	20.08	u	g	0.125	q	768	True	v	1.000	False	1	False	240	g
2	b	20.08	u	g	0.250	q	0	False	v	0.125	False	0	False	200	g
3	b	22.17	u	g	2.250	i	10	False	v	0.125	False	0	False	160	g
4	a	27.25	u	g	0.290	m	108	True	h	0.125	False	1	True	272	g

In [37]:
```
test_data.__eq__('?').sum()
```

```
c:\python\python38\lib\site-packages\pandas\core\ops\array_ops.py:253: FutureWarning: elementwise comparison failed; r
eturning scalar instead, but in the future will perform elementwise comparison
  res_values = method(rvalues)
```

Out[37]:
```
A1     1
A2     0
A3     0
A4     0
A5     0
A6     0
A7     0
A8     0
A9     0
A10    0
A11    0
A12    0
A13    0
A14    0
A15    0
dtype: int64
```

In [38]:
```
#then in the test data set only A1 has missing values
print(test_data['A1'].value_counts())
```

```
b    8
a    5
?    1
Name: A1, dtype: int64
```

In [39]:
```
#replace the missing value with the mode.. here 'b'
test_data['A1'].replace('?','b', inplace=True)
print(test_data['A1'].value_counts())
```

```
b    9
a    5
Name: A1, dtype: int64
```

In [40]:
```
#data types of attributes
print(test_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      14 non-null     object 
 1   A2      14 non-null     float64
 2   A3      14 non-null     object 
 3   A4      14 non-null     object 
 4   A5      14 non-null     float64
 5   A6      14 non-null     object 
 6   A7      14 non-null     int64  
 7   A8      14 non-null     bool   
 8   A9      14 non-null     object 
 9   A10     14 non-null     float64
 10  A11     14 non-null     bool   
 11  A12     14 non-null     int64  
 12  A13     14 non-null     bool   
 13  A14     14 non-null     int64  
 14  A15     14 non-null     object 
dtypes: bool(3), float64(3), int64(3), object(6)
memory usage: 1.5+ KB
None
```

In [41]:
```
#convert data types int to float of some attributes
test_data["A12"]=test_data["A12"].astype(float)
test_data["A7"]=test_data["A7"].astype(float)
test_data["A14"]=test_data["A14"].astype(float)
```

In [42]:
```python
#label encoding of A1 as b=1 and a=0
test_data["A1"]= test_data["A1"].astype('category')
test_data['A1'] = test_data['A1'].cat.codes
test_data.head()
```

Out[42]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
0	1	32.67	y	p	9.000	w	0.0	False	h	5.250	True	0.0	True	154.0	g
1	1	20.08	u	g	0.125	q	768.0	True	v	1.000	False	1.0	False	240.0	g
2	1	20.08	u	g	0.250	q	0.0	False	v	0.125	False	0.0	False	200.0	g
3	1	22.17	u	g	2.250	i	10.0	False	v	0.125	False	0.0	False	160.0	g
4	0	27.25	u	g	0.290	m	108.0	True	h	0.125	False	1.0	True	272.0	g

In [43]:
```python
#one-hot encoding for objects
test_data = pd.get_dummies(test_data, columns=['A3'], prefix=['A3'])
test_data = pd.get_dummies(test_data, columns=['A4'], prefix=['A4'])
test_data = pd.get_dummies(test_data, columns=['A6'], prefix=['A6'])
test_data = pd.get_dummies(test_data, columns=['A9'], prefix=['A9'])
test_data = pd.get_dummies(test_data, columns=['A15'], prefix=['A15'])
```

In [44]:
```python
test_data.head()
```

Out[44]:

	A1	A2	A5	A7	A8	A10	A11	A12	A13	A14	...	A6_c	A6_i	A6_m	A6_q	A6_r	A6_w	A9_bb	A9_h	A9_v	A15_g
0	1	32.67	9.000	0.0	False	5.250	True	0.0	True	154.0	...	0	0	0	0	0	1	0	1	0	1
1	1	20.08	0.125	768.0	True	1.000	False	1.0	False	240.0	...	0	0	0	1	0	0	0	0	1	1
2	1	20.08	0.250	0.0	False	0.125	False	0.0	False	200.0	...	0	0	0	1	0	0	0	0	1	1
3	1	22.17	2.250	10.0	False	0.125	False	0.0	False	160.0	...	0	1	0	0	0	0	0	0	1	1
4	0	27.25	0.290	108.0	True	0.125	False	1.0	True	272.0	...	0	0	1	0	0	0	0	1	0	1

5 rows × 25 columns

In [45]:
```
print(test_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 25 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      14 non-null     int8
 1   A2      14 non-null     float64
 2   A5      14 non-null     float64
 3   A7      14 non-null     float64
 4   A8      14 non-null     bool
 5   A10     14 non-null     float64
 6   A11     14 non-null     bool
 7   A12     14 non-null     float64
 8   A13     14 non-null     bool
 9   A14     14 non-null     float64
 10  A3_u    14 non-null     uint8
 11  A3_y    14 non-null     uint8
 12  A4_g    14 non-null     uint8
 13  A4_p    14 non-null     uint8
 14  A6_aa   14 non-null     uint8
 15  A6_c    14 non-null     uint8
 16  A6_i    14 non-null     uint8
 17  A6_m    14 non-null     uint8
 18  A6_q    14 non-null     uint8
 19  A6_r    14 non-null     uint8
 20  A6_w    14 non-null     uint8
 21  A9_bb   14 non-null     uint8
 22  A9_h    14 non-null     uint8
 23  A9_v    14 non-null     uint8
 24  A15_g   14 non-null     uint8
dtypes: bool(3), float64(6), int8(1), uint8(15)
memory usage: 1.0 KB
None
```

In [46]:
```
#there are missing data columns fro the trained set
# Get missing columns in the training test
missing_cols = set( onehote_data.columns ) - set( test_data.columns )
# Add a missing column in test set with default value equal to 0
for c in missing_cols:
    test_data[c] = 0
# Ensure the order of column in the test set is in the same order than in train set
test_data = test_data[ onehote_data.columns]
```

In [47]:

```
print(test_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 43 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      14 non-null     int8   
 1   A2      14 non-null     float64
 2   A5      14 non-null     float64
 3   A7      14 non-null     float64
 4   A8      14 non-null     bool   
 5   A10     14 non-null     float64
 6   A11     14 non-null     bool   
 7   A12     14 non-null     float64
 8   A13     14 non-null     bool   
 9   A14     14 non-null     float64
 10  A16     14 non-null     int64  
 11  A3_l    14 non-null     int64  
 12  A3_u    14 non-null     uint8  
 13  A3_y    14 non-null     uint8  
 14  A4_g    14 non-null     uint8  
 15  A4_gg   14 non-null     int64  
 16  A4_p    14 non-null     uint8  
 17  A6_aa   14 non-null     uint8  
 18  A6_c    14 non-null     uint8  
 19  A6_cc   14 non-null     int64  
 20  A6_d    14 non-null     int64  
 21  A6_e    14 non-null     int64  
 22  A6_ff   14 non-null     int64  
 23  A6_i    14 non-null     uint8  
 24  A6_j    14 non-null     int64  
 25  A6_k    14 non-null     int64  
 26  A6_m    14 non-null     uint8  
 27  A6_q    14 non-null     uint8  
 28  A6_r    14 non-null     uint8  
 29  A6_w    14 non-null     uint8  
 30  A6_x    14 non-null     int64  
 31  A9_bb   14 non-null     uint8  
 32  A9_dd   14 non-null     int64  
 33  A9_ff   14 non-null     int64  
 34  A9_h    14 non-null     uint8  
 35  A9_j    14 non-null     int64  
 36  A9_n    14 non-null     int64  
 37  A9_o    14 non-null     int64  
 38  A9_v    14 non-null     uint8  
 39  A9_z    14 non-null     int64  
 40  A15_g   14 non-null     uint8  
 41  A15_p   14 non-null     int64  
```

```
 42  A15_s   14 non-null     int64
dtypes: bool(3), float64(6), int64(18), int8(1), uint8(15)
memory usage: 3.0 KB
None
```

In [48]:
```
X_predict = scaler.transform(test_data[feature_names])
```

In [49]:
```
#using linear discriminant analysis 'lda'
y_predict = lda.predict(X_predict)
print(y_predict)
```

```
['Success' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure' 'Failure'
 'Failure' 'Failure' 'Failure' 'Failure' 'Success' 'Success' 'Success']
```

In [50]:
```
original_test_data.insert(15,"A16",y_predict,True)
original_test_data.head(14)
```

Out[50]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	32.67	y	p	9.000	w	0	False	h	5.250	True	0	True	154	g	Success
1	?	20.08	u	g	0.125	q	768	True	v	1.000	False	1	False	240	g	Failure
2	b	20.08	u	g	0.250	q	0	False	v	0.125	False	0	False	200	g	Failure
3	b	22.17	u	g	2.250	i	10	False	v	0.125	False	0	False	160	g	Failure
4	a	27.25	u	g	0.290	m	108	True	h	0.125	False	1	True	272	g	Failure
5	b	31.58	y	p	0.750	aa	0	False	v	3.500	False	0	True	320	g	Failure
6	a	20.83	u	g	8.500	c	351	False	v	0.165	False	0	False	0	g	Failure
7	b	48.08	u	g	3.750	i	2	False	bb	1.000	False	0	False	100	g	Failure
8	b	29.83	u	g	3.500	c	0	False	v	0.165	False	0	False	216	g	Failure
9	a	41.58	u	g	1.040	aa	237	False	v	0.665	False	0	False	240	g	Failure
10	b	33.17	u	g	1.040	r	31285	False	h	6.500	True	0	True	164	g	Failure
11	a	18.92	u	g	9.000	aa	591	True	v	0.750	True	2	False	88	g	Success
12	a	24.75	u	g	3.000	q	500	True	h	1.835	True	19	False	0	g	Success
13	b	21.00	y	p	4.790	w	300	True	v	2.250	True	1	True	80	g	Success

In [ ]: